arXiv:1302.2184v1 [cs.DS] 9 Feb 2013

The two-edge connectivity survivable-network design problem in planar graphs

Glencora Borradaile
Oregon State University

Philip Klein
Brown University

February 12, 2013

Abstract

Consider the following problem: given a graph with edge costs and a subset Q of vertices, find a minimum-cost subgraph in which there are two edge-disjoint paths connecting every pair of vertices in Q. The problem is a failure-resilient analog of the Steiner tree problem arising, for example, in telecommunications applications. We study a more general mixed-connectivity formulation, also employed in telecommunications optimization. Given a number (or *requirement*) $r(v) \in \{0, 1, 2\}$ for each vertex v in the graph, find a minimum-cost subgraph in which there are $\min\{r(u), r(v)\}$ edge-disjoint u-to-v paths for every pair u, v of vertices.

We address the problem in planar graphs, considering a popular relaxation in which the solution is allowed to use multiple copies of the input-graph edges (paying separately for each copy). The problem is max SNP-hard in general graphs and strongly NP-hard in planar graphs. We give the first polynomial-time approximation scheme in planar graphs. The running time is $O(n \log n)$.

Under the additional restriction that the requirements are only non-zero for vertices on the boundary of a single face of a planar graph, we give a polynomial-time algorithm to find the optimal solution.

1 Introduction

In the field of telecommunications network design, an important requirement of networks is resilience to link failures [25]. The goal of the survivable network problem is to find a graph that provides multiple routes between pairs of terminals. In this work we address a problem concerning edge-disjoint paths. For a set S of non-negative integers, an instance of the S*-edge connectivity design* problem is a pair $(G, \boldsymbol{r})$ where $G = (V, E)$ is a undirected graph with edge costs $\boldsymbol{c} : V \to \Re^+$ and *connectivity requirements* $\boldsymbol{r} : V \to S$. The goal is to find a minimum-cost subgraph of G that, for each pair u, v of vertices, contains at least $\min\{\boldsymbol{r}(u), \boldsymbol{r}(v)\}$ edge-disjoint u-to-v paths.

In telecommunication-network design, failures are rare; for this reason, there has been much research on *low-connectivity network design* problems, in which the maximum connectivity requirement is two. Resende and Pardalos [25] survey the literature, which includes heuristics, structural results, polyhedral results, computational results using cutting planes, and approximation algorithms. This work focuses on $\{0, 1, 2\}$-edge connectivity problems in planar graphs.

We consider the previously studied variant wherein the solution subgraph is allowed to contain multiple copies of each edge of the input graph (a *multi*-subgraph); the costs of the edges in the solution are counted according to multiplicity. For $\{0, 1, 2\}$-connectivity, at most two copies of an

edge are needed. We call this the *relaxed* version of the problem and use the term *strict* to refer to the version of the problem in which multiple copies of edges of the input graph are disallowed.

A *polynomial-time approximation scheme* (PTAS) for an optimization problem is an algorithm that, given a fixed constant $\epsilon > 0$, runs in polynomial time and returns a solution within $1 + \epsilon$ of optimal. The algorithm's running time need not be polynomial in ϵ. The PTAS is *efficient* if the running time is bounded by a polynomial whose degree is independent of ϵ. In this paper, we focus on designing a PTAS for $\{0, 1, 2\}$-edge connectivity.

Two edge-connected spanning subgraphs A special case that has received much attention is the problem of finding a minimum-cost subgraph of G in which *every* pair of vertices is two edge-connected. Formally this is the strict $\{2\}$-edge connectivity design problem. This problem is NP-hard [12] (even in planar graphs, by a reduction from Hamiltonian cycle) and max-SNP hard [10] in general graphs. In general graphs, Frederickson and JáJá [13] gave an approximation ratio of 3 which was later improved to 2 (and 1.5 for unit-cost graphs) by Khuller and Vishkin [18] and 5/4 by Jothi, Raghavachari, and Varadarajan [17]. In planar graphs, Berger et al. [4] gave a polynomial-time approximation scheme (PTAS) for the relaxed $\{1, 2\}$-edge connectivity design problem and Berger and Grigni [5] gave a PTAS for the strict $\{2\}$-edge connectivity design problem. Neither of these algorithms is efficient; the degree of the polynomial bounding the running time grows with $1/\epsilon$. For the relaxed version of the problem, the techniques of Klein [20] can be used to obtain a linear-time approximation scheme.

Beyond spanning When a vertex can be assigned a requirement of zero, edge-connectivity design problems include the *Steiner tree problem*: given a graph with edge costs and given a subset of vertices (called the *terminals*), find a minimum-cost connected subgraph that includes all vertices in the subset. More generally, we refer to any vertex with a non-zer connectivity requirement as a terminal. For $\{0, 2\}$-edge connectivity design problem, in general graphs, Ravi [24] showed that Frederickson and JáJá's approach could be generalized to give a 3-approximation algorithm (in general graphs). Klein and Ravi [21] gave a 2-approximation for the $\{0, 1, 2\}$-edge connectivity design problem. (In fact, they solve the even more general version in which requirements $\boldsymbol{r}(u, v)$ are specified for *pairs* u, v of vertices.) This result was generalized to connectivity requirements higher than two by Williamson et al. [27], Goemans et al. [14], and Jain [16]. These algorithms each handle the strict version of the problem.

In their recent paper on the spanning case [5], Berger and Grigni raise the question of whether there is a PTAS for the $\{0, 2\}$-edge connectivity design problem in planar graphs. In this paper, we answer that question in the affirmative for the relaxed version. The question in the case of the strict version is still open.

1.1 Summary of new results

Our main result is a PTAS for the relaxed $\{0, 1, 2\}$-edge connectivity problem in planar graphs:

Theorem 1.1. *For any ϵ, there is an $O(n \log n)$ algorithm that, given a planar instance $(G, \boldsymbol{r})$ of relaxed $\{0, 1, 2\}$-edge connectivity, finds a solution whose cost is at most $1 + \epsilon$ times optimal.*

This result builds on the work of Borradaile, Klein and Mathieu [7, 8, 9] which gives a PTAS for the Steiner tree (i.e. $\{0,1\}$-edge connectivity) problem. This is the first PTAS for a non-spanning two-edge connectivity problem in planar graphs.

Additionally, we give an exact, polynomial-time algorithm for the special case where are the vertices with non-zero requirement are on the boundary of a common face:

Theorem 1.2. *There is an $O(k^3 n)$-time algorithm that finds an optimal solution to any planar instance $(G, \boldsymbol{r})$ of relaxed $\{0,1,2\}$-edge-connectivity in which only k vertices are assigned nonzero requirements and all of them are on the boundary of a single face. For instances of relaxed $\{0,2\}$-edge connectivity (i.e. all requirements are 0 or 2), the algorithm runs in linear time.*

1.2 Organization

We start by proving Theorem 1.2 in Section 3. The proof of this result is less involved and provides a good warm-up for the proof of Theorem 1.1. The algorithm uses the linear-time shortest-path algorithm for planar graphs [15] and a polynomial-time algorithm for the equivalent boundary Steiner-tree problem [11] as black boxes.

In order to prove Theorem 1.1, we need to review the framework developed for the Steiner tree problem in planar graphs. We give an overview of this framework in Section 4 and show how to use it to solve the relaxed $\{0,1,2\}$-edge connectivity problem. The correctness of the PTAS relies on a Structure Theorem (Theorem 4.6) which bounds the number of interactions of a solution between different regions of the graph while paying only a small relative penalty in cost. We prove this Structure Theorem in Section 5. The algorithm itself requires a dynamic program; we give the details for this in Section 6.

2 Basics

We consider graphs and multi-subgraphs. A multi-subgraph is a subgraph where edges may be included with multiplicity. In proving the Structure Theorem we will replace subgraphs of a solution with other subgraphs. In doing so, two of the newly introduced subgraphs may share an edge. We assume that we do so by introducing multiple copies of that edge. It is convenient to view two copies of an edge as being embedded next to eachother, forming a cycle.

For a subgraph H of a graph G, we use $V(H)$ to denote the set of vertices in H. For a graph G and set of edges E, G/E denotes the graph obtained by contracting the edges E.

For a path P, $P[x,y]$ denotes the x-to-y subpath of P for vertices x and y of P; $\mathrm{end}(P)$ and $\mathrm{start}(P)$ denote the first and last vertices of P; $\mathrm{rev}(P)$ denotes the reverse of path P. For paths A and B, $A \circ B$ denotes the concatenation of A and B. See Figure 1 for an illustration of the notion of paths crossing. A cycle is non-self-crossing if every pair of subpaths of the cycle do not cross.

We employ the usual definitions of planar embedded graphs. For a face f, the cycle of edges making up the boundary of f is denoted ∂f. We assume the planar graph G is connected and is embedded in the plane, so there is a single infinite face, and we denote its boundary by ∂G.

For a cycle C in a planar embedded graph, $C[x,y]$ denotes an x-to-y path in C for vertices x and y of C. There are two such paths and the choice between the two possibilities will be disambiguated by always choosing the subpath in the clockwise direction. A cycle C is said to *enclose* the faces that are embedded inside it. C encloses an edge/vertex if the edge/vertex is embedded inside it or

on it. In the former case, C *strictly encloses* the edge/vertex. For non-crossing x-to-y paths P and Q, P is said to be left of Q if $P \circ \mathrm{rev}(Q)$ is a clockwise cycle.

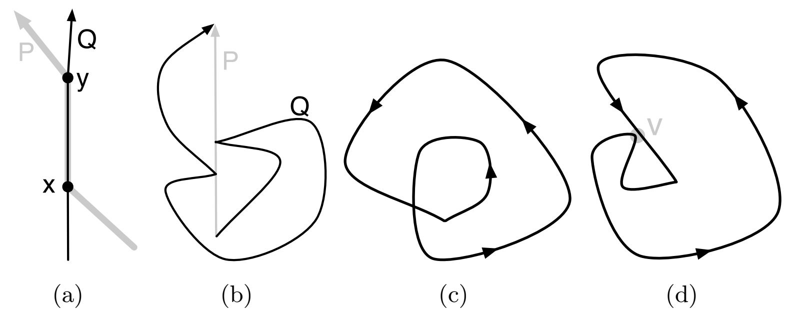


Figure 1: (a) P crosses Q. (b) P and Q are noncrossing; Q is left of P (c) A self-crossing cycle. (d) A non-self-crossing cycle (non-self-crossing allows for repeated vertices, i.e. v.)

We will use the following as a subroutine:

Theorem 2.1. *Erickson, Monma and Veinott* **[11]** *Let G be a planar embedded graph with edge-costs and let Q be a set of k terminals that all lie on the boundary of a single face. Then there is an algorithm to find an minimum-cost Steiner tree of G spanning Q in time $O(nk^3 + (n \log n)k^2)$ or $O(nk^3)$ time using the algorithm of [15].*

2.1 Edge-connectivity basics

Since we are only interested in connectivity up to and including two-edge connectivity, we define the following: For a graph H and vertices x, y, let

$$c_H(x, y) = \min\{2, \text{maximum number of edge-disjoint } x\text{-to-}y \text{ paths in } H\}.$$

For two multi-subgraphs H and H' of a common graph G and for a subset S of the vertices of G, we say H' *achieves the two-connectivity of* H for S if $c_{H'}(x, y) \geq c_H(x, y)$ for every $x, y \in S$. We say H' achieves the *boundary* two-connectivity of H if it achieves the two-connectivity of H for $S = V(\partial G)$.

Several of the results in the paper build on observations of the structural property of two-edge connected graphs. The first is a well-known property:

Lemma 2.2 (Transitivity)**.** *For any graph H, for vertices $u, v, w \in V(H)$, $c_H(u, w) \geq \min\{c_H(u, v), c_H(v, w)\}$*

Note that in the following, we can replace "strictly enclose" with "does not enclose" without loss of generality (by viewing a face enclosed by C as the infinite face).

Lemma 2.3 (Empty Cycle)**.** *Let H be a (multi-)subgraph of G and let C be a non-self-crossing cycle of H that strictly encloses no terminals. Let H' be the subgraph of H obtained by removing the edges of H that are strictly enclosed by C. Then H' achieves the two-connectivity of H.*

Proof. See Figure 2(b). Without loss of generality, view C as a clockwise cycle. Consider two terminals x and y. We show that there are $c_H(x, y)$ edge-disjoint x-to-y paths in H that do not use edges strictly enclosed by C. There are two nontrivial cases:

$c_H(x, y) = 1$: Let P be an x-to-y path in H. If P intersects C, let x_P be the first vertex of P that is in C and let y_P be the last vertex of P that is in C. Let $P' = P[x, x_P] \circ C[x_P, y_P] \circ P[y_P, y]$. If P does not intersect C, let $P' = P$. P' is an x-to-y path in H that has no edge strictly enclosed by C.

$c_H(x, y) = 2$: Let P and Q be edge-disjoint x-to-y paths in H. If Q does not intersect C, then P' and Q are edge-disjoint paths, neither of which has an edge strictly enclosed by C (where P' is as defined above). Suppose that both P and Q intersect C. Define x_Q and y_Q as for P. Suppose these vertices are ordered x_P, x_Q, y_Q, y_P around C. Then $P[x, x_P] \circ C[x_P, y_Q] \circ Q[y_Q, y]$ and $Q[x, x_Q] \circ \text{rev}(C[y_P, x_Q]) \circ P[y_P, y]$ are edge disjoint x-to-y paths that do not use any edges enclosed by C. This case is illustrated in Figure 2; other cases (for other orderings of $\{x_P, x_Q, y_Q, y_P\}$ along C) follow similarly.

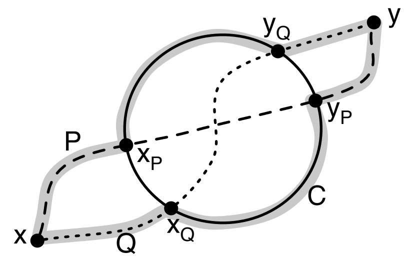


Figure 2: An illustration of the proof of Lemma 2.3: there are edge-disjoint x-to-y paths (grey) that do not use edges enclosed by C.

We have shown that we can achieve the boundary two-connectivity of H without using any edges strictly enclosed by a cycle of H. The lemma follows. □

2.2 Vertex-connectivity basics

The observations in this section do not involve planarity. Although our results are for edge connectivity, we use vertex connectivity in Section 5 to simplify our proofs.

Vertices x and y are *biconnected* (a.k.a. two-vertex-connected) in a graph H if H contains two x-to-y paths that do not share any internal vertices. For a subset S of vertices of H, we say H is *S-biconnected* if for every pair x, y of vertices of S, H contains a simple cycle through x and y. We refer to the vertices of S as *terminals*.

Lemma 2.4. *A minimal S-biconnected graph is biconnected.*

Proof. Suppose H has a cut-vertex v: $H = H_1 \cup H_2$ where $H_1 \cap H_2 = \{v\}$. If H_1 and H_2 both have terminals then H does not biconnect every pair of terminals. If, say, H_2 does not have a terminal then $H - (H_2 - \{v\})$ is a smaller subgraph that biconnects the terminals. □

Theorem 2.5. *Let H be a minimal S-biconnected graph. Every cycle in H contains a vertex of S.*

Proof. Assume for a contradiction that H contains a cycle C that does not contain any terminals.

We say vertices x and y of C form a *connected pair* if there is an x-to-y path whose internal vertices do not intersect C. Since C does not contain any terminals, there must be a terminal t

not in C. Since H is two-vertex-connected (Lemma 2.4), t must have two paths to C, so there must be a connected pair on C. Let P be a minimal nonempty subpath of C whose endpoints are a connected pair and such that no other pair of vertices of P form a connected pair.

By minimality of P we get:

Observation 2.6. *No connected component of $H - C$ is adjacent in H to an internal vertex of P.*

Let H' be the graph obtained by deleting the edges and internal vertices of P. Since C contains no terminals, H' contains all the terminals of H. By Observation 2.6, the only edges deleted are those of P. Let $Q_1 = C - P$. By construction, Q_1 is in H'. Since x and y are a connected pair, there is a path Q_2 in H' connecting x and y that do not share any internal vertices with C. Therefore $Q_1 \cup Q_2$ is a simple cycle. We get:

Observation 2.7. *Vertices x and y are biconnected in H'.*

Let w be any vertex in H'. We show that w is not a cut vertex, implying that H' is biconnected and contradicting the minimality of H.

By Observation 2.7, there is an x-to-y path Q in $H' - \{w\}$. Let z and z' be any vertices in $H' - \{w\}$. Since H is biconnected, there is a simple z-to-z' path R in G that avoids w. If R does not use any vertex of P then R is a path in $H' - w$ (and so w is not a cut vertex). If R uses some edge of P, by Observation 2.6, R uses all of P. Then $(R - P) \cup Q$ is a z-to-z' path in $H' - w$. $\square$

Theorem 2.8. *Let H be a minimal S-biconnected graph. For any cycle C in H, every C-to-C path contains a vertex of S.*

Proof. By Lemma 2.4, H is biconnected. Let P be a minimal C-to-C path. Assume for a contradiction that P does not include a vertex of S. Write $C = c_1 \circ C_2$ where the endpoints of C_1 and C_2 coincide with the endpoints of P. Consider a simple path Q in $H - (C_1 \cup P)$ that starts and ends on P but whose internal vertices are not on P and chosen to minimize the cost of the subpath P' between the endpoints of Q. (Such a path exists, namely C_2.) Let e be the first edge of P'. (See Figure 3.) Let B be the biconnected component of $H - e$ that includes C. (It also includes Q and the edges of $P - P'$.) We claim that B includes all of H except P'. Assume not; then there must be some connected component K consisting of other biconnected components of $H - e$. Since H is biconnected, K must share two vertices with $B \cup P'$. If the two vertices are on $P' - e$, this would contradict the choice of Q. If the two vertices are anywhere else, some biconnected component in K would be biconnected with B in $H - e$, a contradiction. $\square$

3 An exact algorithm for boundary $\{0, 1, 2\}$-edge connectivity

Our algorithm for the boundary case of $\{0, 1, 2\}$-edge connectivity, as formalized in Theorem 1.2 is based on the observation that there is an optimal solution to the problem that is the union of Steiner trees whose terminals are boundary vertices, allowing us to employ the boundary-Steiner-tree algorithm of Theorem 2.1.

When terminals are restricted to the boundary of the graph, no cycle can strictly enclose a terminal. By Lemma 2.3, we get:

Corollary 3.1. *Let H be a subgraph of G and let H' be a minimal subgraph of H that achieves the boundary two-connectivity of H. Then in H' every cycle C strictly encloses no edges.*

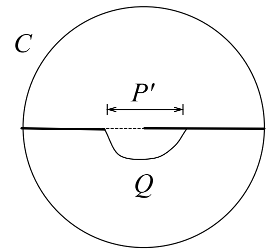


Figure 3: Part of the minimal S-biconnected graph H in the proof of Theorem 2.8: the cycle C, the C-to-C path P, the P-to-P path Q, and the subpath P' of P between the endpoints of Q. When the edge e (indicated by the dotted line) is removed, there remains a biconnected component that includes C, Q, and the edges of P not in P', along with other parts of H.

In the following we will assume that the boundary of the graph G is a simple cycle; that is, a vertex appears at most once along ∂G. Let us see why this is a safe assumption. Suppose the boundary of G is not simple: there is a vertex v that appears at least twice along ∂G. Partition G into two graphs G_1 and G_2 such that v appears exactly once along ∂G_1 and $E(\partial G) = E(\partial G_1) \cup E(\partial G_2)$. Let x be a vertex of ∂G_1 and let y be a vertex of ∂G_2. Then $c_G(x, y) = \min\{c_{G_1}(x, v), c_{G_2}(v, y)\}$, allowing us to define new connectivity requirements and solve the problem separately for G_1 and G_2.

Lemma 3.2. *Let P and Q be leftmost non-self-crossing x_P-to-y_P and x_Q-to-y_Q paths, respectively, where x_P, y_P, x_Q, and y_Q are vertices in clockwise order on ∂G. Then P does not cross Q.*

Proof. For a contradiction, assume that Q crosses P. Refer to Figure 4(a). Let C (interior shaded) be the cycle $P \circ \text{rev}(\partial G[x_P, y_P])$. C strictly encloses neither x_Q nor y_Q. If Q crosses P, there must be a subpath of Q enclosed by C. Let x be the first vertex of Q in P and let y be the last. There are two cases:

$x \in P[y, y_P]$: Refer to Figure 4(a). In this case, $\text{rev}(P[y, x])$ is left of $Q[x, y]$ and so $Q[x_Q, x] \circ \text{rev}(P[y, x]) \circ Q[y, y_Q]$ (grey path) is left of Q, contradicting the leftmostness of Q.

$x \in P[x_P, y]$: Refer to Figure 4(b). In this case, $Q[x_Q, x] \circ \text{rev}(P[x_P, x]) \circ \partial G[x_P, x_Q]$ (shaded interior) is a cycle that strictly encloses y and does not enclose y_Q. Since y is the last vertex of Q on P, Q must cross itself, a contradiction. □

Lemma 3.3. *Let H be a subgraph of G. Let S be a subset of $V(\partial G)$ such that, for every $x, y \in S$, $c_H(x, y) = 2$. Then there is a non-self-crossing cycle C in H such that $S \subseteq V(C)$ and the order that C visits the vertices in S is the same as their order along ∂G.*

Proof. Assume that the vertices of S are in the clockwise order $s_0, s_1, \ldots, s_{k-1}$ along ∂G.

Let P_i be the leftmost non-self-crossing s_{i-1}-to-s_i path in H, taking the indices modulo k. Let $C = P_1 \circ P_2 \circ \cdots \circ P_{k-1}$. Certainly C visits each of the vertices $s_0, s_1, \ldots$ in order. By Lemma 3.2, P_i does not cross P_j for all $i \neq j$. Therefore, C is non-self-crossing, proving the lemma. □

We now give an algorithm for the following problem: given a planar graph G with edge costs and an assignment $\boldsymbol{r}$ of requirements such that $r(v) > 0$ only for vertices v of ∂G, find a minimum-cost multi-subgraph H of G that satisfies the requirements (i.e. such that there are at least $\min\{r(x), r(y)\}$ edge-disjoint x-to-y paths in H).

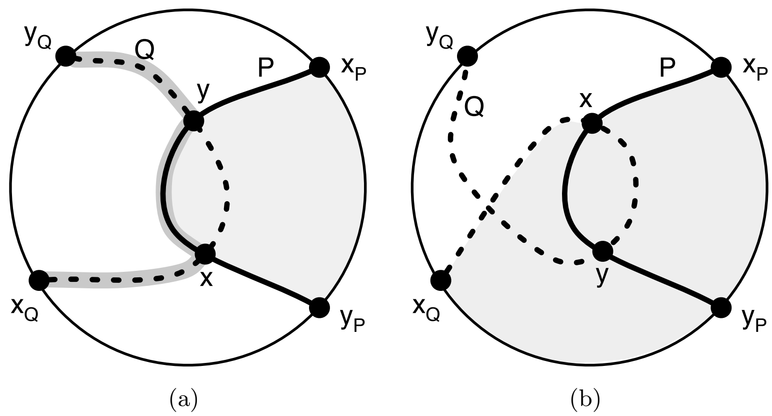


Figure 4: Illustration of Lemma 3.2: there exist leftmost paths that do not cross.

BOUNDARY2EC(G, r)

1. Let $q_1, q_2, \ldots$ be the cyclic ordering of vertices $\{v \in V(\partial G) : \boldsymbol{r}(v) = 2\}$.
2. For $i = 1, \ldots$, let $X_i = \{q_i\} \cup \{v \in V(\partial G[q_i, q_{i+1}]) \ : \ \boldsymbol{r}(v) = 1\} \cup \{q_{i+1}\}$.
3. For $i = 1, \ldots$, let T_i be the minimum-cost Steiner tree spanning X_i.
4. Return the disjoint union $\cup_i T_i$.

We show that BOUNDARY2EC correctly finds the minimum-cost multi-subgraph of G satisfying the requirements. Let OPT denote an optimal solution. By Lemma 3.3, OPT contains a non-self-crossing cycle C that visits $q_1, q_2, \ldots$ (as defined in BOUNDARY2EC). By Corollary 3.1, C strictly encloses no edges of OPT. Let P_i be the leftmost q_i-to-q_{i+1} path in C. The vertices in X_i are connected in OPT, by the input requirements. Let S_i be the subgraph of OPT that connects X_i. This subgraph is enclosed by $\partial G[q_i, q_{i+1}] \circ C[q_i, q_{i+1}]$. Replacing S_i by T_i achieves the same connectivity among vertices v with $\boldsymbol{r}(v) > 0$ without increasing the cost.

We will use the following lemma to give an efficient implementation of BOUNDARY2EC.

Lemma 3.4. *Let a, b and c be vertices ordered along the clockwise boundary ∂G of a planar graph G. Let T_a be the shortest-path tree rooted at a (using edge costs for lengths). Then for any set of terminals Q in $\partial G[b, c]$, there is a minimum-cost Steiner tree connecting them that enclosed by the cycle $\partial G[b, c] \circ T_a[c, b]$.*

Proof. Refer to Figure 5. Let $C = \partial G[b, c] \circ T_a[c, b]$. Let T be a minimum-cost Steiner tree in G connecting Q. Suppose some part of T is not enclosed by C. Let T' be a maximal subtree of T not enclosed by C. The leaves of T' are on $T_a[c, b]$. Let P be the minimum subpath of $T_a[b, c]$ that spans these leaves. Let P' be the start(P)-to-end(P) path in T'. See Figure 5.

We consider the case when start(P') is a vertex of $T_a[a, b]$ and end(P') is a vertex of $T_a[a, c]$ (the other cases, when start(P') and end(P') are either both vertices of $T_a[a, b]$ or both vertices of $T_b[a, c]$, are simpler). Then P' must cross $T_a[a, x]$ where x is the last vertex common to $T_a[a, b]$ and $T_a[a, c]$ (i.e. the lowest common ancestor in T_a of b and c). Let y be a vertex of $P' \cap T_a[a, x]$. Since T_a is a shortest-path tree in an undirected path, every subpath of $T_a[a, z]$ and $T_a[z, a]$, for

any vertex z, is a shortest path. We have that:

$$\begin{aligned}
\boldsymbol{c}(P') &= \boldsymbol{c}(P'[\text{start}(P'), y]) + \boldsymbol{c}(P'[y, \text{end}(P')]) \\
&\geq \boldsymbol{c}(T_a[\text{start}(P'), y]) + \boldsymbol{c}(T_a[y, \text{end}(P')]) \\
&\geq \boldsymbol{c}(T_a[\text{start}(P'), \text{end}(P')]) \\
&\geq \boldsymbol{c}(P)
\end{aligned}$$

Let $\widehat{T} = T - T' \cup P$. By construction, $\widehat{T}$ spans Q. Using that $\boldsymbol{c}(P') \geq \boldsymbol{c}(P)$, we have that $\boldsymbol{c}(\widehat{T}) = \boldsymbol{c}(T) - \boldsymbol{c}(T') + \boldsymbol{c}(P) \leq \boldsymbol{c}(T) - \boldsymbol{c}(T') + \boldsymbol{c}(P') \leq \boldsymbol{c}(T)$ since P' is a subpath of T'.

Repeating this process for every subtree of T not enclosed by C results in a tree enclosed by C spanning Q that is no longer than T. $\square$

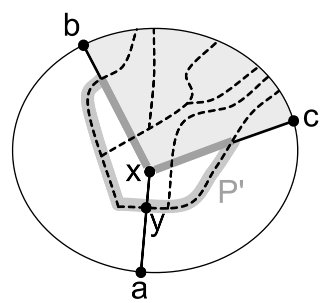


Figure 5: There is a tree $\widehat{T}$ that is just as cheap as T (dotted) and spans the terminals between b and c but is enclosed by C (whose interior is shaded). $\widehat{T}$ is composed of the portion of T enclosed by C plus P, the thick grey path.

We describe an $O(k^3 n)$-time implementation of Boundary2EC (where k is the number of terminals). Compute a shortest-path tree T rooted at terminal q_1 in linear time. For each i, consider the graph G_i enclosed by $C_i = \partial G[q_i, q_{i+1}] \circ T[q_{i+1}, q_i]$. Compute the minimum Steiner tree spanning X_i in G_i. By Lemma 3.4, T_i has the same cost as the minimum spanning tree spanning X_i in G. Since each edge of G appears in at most two subgraphs G_i and G_j, the trees T_i can be computed in $O(k^3 n)$ time (by Theorem 2.1).

Note: if the requirements are such that $\boldsymbol{r}(v) \in \{0, 2\}$ for every vertex v on the boundary of G, then the sets X_i have cardinality 2. Instead of computing Steiner trees in Step 3, we need only compute shortest paths. The running time for this special case is therefore linear.

This completes the proof of Theorem 1.2.

4 A PTAS framework for connectivity problems in planar graphs

In this section, we review the approach used in [9] to give a PTAS *framework* for the Steiner tree problem in planar graphs. While the contents of this section are largely a summary of the framework, we generalize where necessary for the survivable network design problem, but refer the reader to the original paper [9] for proofs and construction details that are not unique to the focus of this article.

Herein, denote the set of terminals by Q. OPT denotes an optimal solution to the survivable network design problem. We overload this notation to also represent the cost of the optimal solution.

4.1 Mortar graph and bricks

The framework relies on an algorithm for finding a subgraph MG of G, called the *mortar graph* [9]. The mortar graph spans Q and has total cost no more than $9\epsilon^{-1}$ times the cost of a minimum Steiner tree in G spanning Q (Lemma 6.9 of [9]). Since a solution to the survivable network problem necessarily spans Q, the mortar graph has cost at most

$$9\epsilon^{-1} \cdot \mathrm{OPT}. \tag{1}$$

The algorithm for computing MG first computes a 2-approximate Steiner tree [22, 26, 28] and then augments this subgraph with short paths. The resulting graph is a grid-like subgraph (the bold edges in Figure 6(a)) many of whose subpaths are ϵ-short:

Definition 4.1. *A path P in a graph G is ϵ-short if for every pair of vertices x and y on P,*

$$dist_P(x, y) \leq (1 + \epsilon) dist_G(x, y).$$

That is, the distance from x to y along P is at most $(1 + \epsilon)$ times the distance from x to y in G.

For each face f of the mortar graph, the subgraph of G enclosed by that face (including the edges and vertices of the face boundary) is called a *brick* (Figure 6(b)), and the brick's boundary is defined to be f. The boundary of a brick B is written ∂B. The *interior of B* is defined to be the subgraph of edges of B not belonging to ∂B. The interior of B is written $\mathrm{int}(B)$.

Bricks satisfy the following:

Lemma 4.2 (Lemma 6.10 [9])**.** *The boundary of a brick B, in counterclockwise order, is the concatenation of four paths W_B, S_B, E_B, N_B (west, south, east, north) such that:*

1. *The set of edges $B - \partial B$ is nonempty.*
2. *Every vertex of $Q \cap B$ is in N_B or in S_B.*
3. *N_B is 0-short in B, and every proper subpath of S_B is ϵ-short in B.*
4. *There exists a number $t \leq \kappa(\epsilon)$ and vertices $s_0, s_1, s_2, \ldots, s_t$ ordered from west to east along S_B such that, for any vertex x of $S_B[s_i, s_{i+1})$, the distance from x to s_i along S_B is less than ϵ times the distance from x to N_B in B: $dist_{S_B}(x, s_i) < \epsilon\, dist_B(x, N_B)$.*

The number $\kappa(\epsilon)$ is given by:

$$\kappa(\epsilon) = 4\epsilon^2(1 + \epsilon^1) \tag{2}$$

The mortar graph has some additional properties. Let B_1 be a brick, and suppose B_1's eastern boundary E_{B_1} contains at least one edge. Then there is another brick B_2 whose western boundary W_{B_2} exactly coincides with E_{B_1}. Similarly, if B_2 is a brick whose western boundary contains at least one edge then there is a brick B_1 whose eastern boundary coincides with B_2's western boundary.

The paths forming eastern and western boundaries of bricks are called *supercolumns*.

Lemma 4.3 (Lemma 6.6 [9])**.** *The sum of the costs of the edges in supercolumns is at most $\epsilon\,\mathrm{OPT}$.*

The mortar graph and the bricks are building blocks of the structural properties required for designing an approximation scheme. Borradaile, Klein and Mathieu demonstrated that there is a near-optimal Steiner tree whose interaction with the mortar graph is "simple" [9]. We prove a similar theorem in Section 5. In order to formalize the notion of "simple", we select a subset of vertices on the boundary of each brick, called *portals*, and define a *portal-connected graph.*

4.2 Portals and simple connections

We define a subset of θ evenly spaced vertices along the boundary of every brick. The value of θ depends polynomially on the precision and the number of interactions, α, a solution must be allowed between bricks to achieve that precision; α in turn depends polynomially on ϵ and is defined by Equation (6).

$$\theta(\epsilon) \text{ is } O(\epsilon^{-2}\alpha(\epsilon)) \tag{3}$$

The portals are selected to satisfy the following:

Lemma 4.4 (Lemma 7.1 [9])**.** *For any vertex x on ∂B, there is a portal y such that the cost of the x-to-y subpath of ∂B is at most $1/\theta$ times the cost of ∂B.*

Recall that, for each face f of the mortar graph MG, there is a corresponding brick B, and that B includes the vertices and edges comprising the boundary of f. The next graph construction starts with the disjoint union of the mortar graph MG with all the bricks. Each edge e of the mortar graph is represented by three edges in the disjoint union: one in MG (the *mortar-graph copy* of e) and one on each of two brick boundaries (the *brick copies* of e). Similarly, each vertex on the boundary of a brick occurs several times in the disjoint union.

The *portal-connected graph*, denoted $\mathcal{B}^+(MG)$, is obtained from the disjoint union as follows: a copy of each brick B of MG is embedded in the interior of the corresponding face of MG, and each portal p of B is connected by an artificial edge to the corresponding vertex of MG. The construction is illustrated in Figure 6(c). The artificial edges are called *portal edges*, and are assigned zero cost.

Noting that each vertex v on the boundary of a brick occurs several times in $\mathcal{B}^+(MG)$, we identify the original vertex v of G with that duplicate in $\mathcal{B}^+(MG)$ that belongs to MG. In particular, each terminal (vertex in Q) is considered to appear exactly once in $\mathcal{B}^+(MG)$, namely in MG. Thus the original instance gives rise to an instance in $\mathcal{B}^+(MG)$: the goal is to compute the optimal solution w.r.t. the terminals on MG in $\mathcal{B}^+(MG)$ and then map the edges of this solution to G. Since G can be obtained from $\mathcal{B}^+(MG)$ by contracting portal edges and identifying all duplicates of each edge and all duplicates of each vertex, we infer:

Lemma 4.5. *Let H be a subgraph of $\mathcal{B}^+(MG)$ that, for each pair of terminals u, v, contains at least $\min\{r(u), r(v)\}$ edge-disjoint u-to-v paths. Then the subgraph of G consisting of edges of H that are in G has the same property.*

The graph $\mathcal{B}^\div(MG)$ is obtained by contracting each brick in $\mathcal{B}^+(MG)$ to a single vertex, called a *brick vertex*, as illustrated in Figure 6(d). This graph will be used in designing the dynamic program in Section 6.

4.3 Structure Theorem

Lemma 4.5 implies that, to find an approximately optimal solution in G, it suffices to find a solution in $\mathcal{B}^+(MG)$ whose cost is not much more than the cost of the optimal solution in G. The following theorem, which we prove in Section 5, suggests that this goal is achievable. An equivalent theorem was proven for the Steiner tree problem [9].

Theorem 4.6 (Structure Theorem)**.** *For any $\epsilon > 0$ and any planar instance $(G, \boldsymbol{r})$ of the $\{0, 1, 2\}$-edge connectivity problem, there exists a feasible solution S to the corresponding instance $(\mathcal{B}^+(MG), \boldsymbol{r})$ such that*

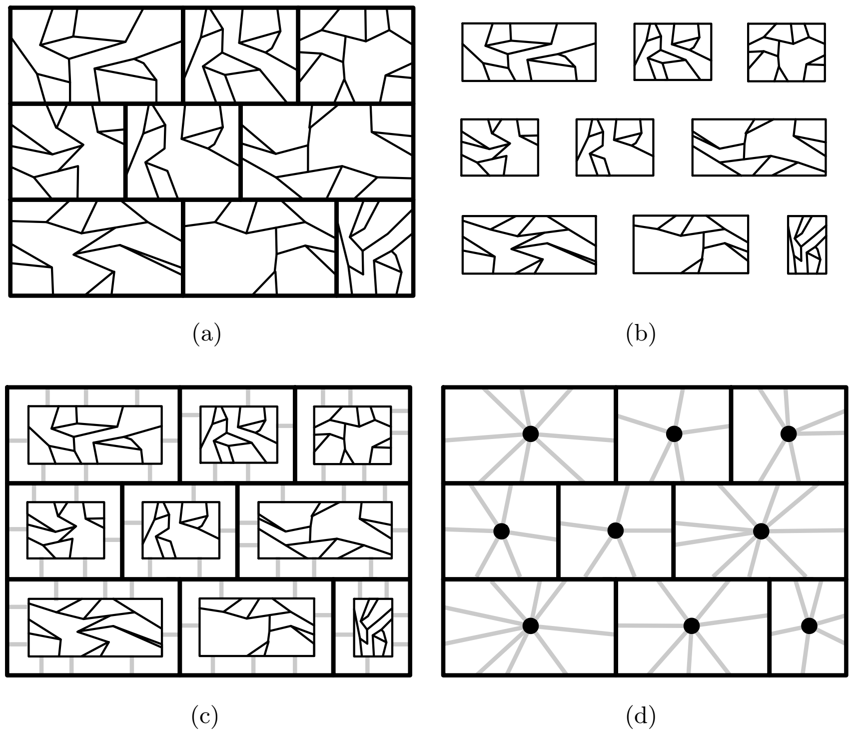


Figure 6: (a) The mortar graph in bold, (b) the set of bricks, (c) the portal-connected graph $\mathcal{B}^{+}(MG)$, and (d) the brick contracted graph $\mathcal{B}^{\div}(MG)$.

- *the cost of S is at most $(1+c\epsilon)$OPT where c is an absolute constant, and*
- *the intersection of S with any brick B is the union of a set of non-crossing trees whose leaves are portals.*

4.4 Approximation scheme

The approximation scheme consists of the following steps.

Step 1: Find the mortar graph MG.

Step 2: Decompose MG into "parcels", subgraphs with the following properties:

(a) Each parcel consists of the boundaries of a disjoint set of faces of MG. Since each edge of MG belongs to the boundaries of exactly two faces, it follows that each edge belongs to at most two parcels.

(b) The cost of all boundary edges (those edges belonging to two parcels) is at most $\frac{1}{\eta}\boldsymbol{c}(MG)$. We choose η so that this bound is $\frac{\epsilon}{2}\boldsymbol{c}(OPT)$:

$$\eta = \eta(\epsilon) = \lceil 20\epsilon^{-2} \rceil \tag{4}$$

(c) The planar dual of each parcel has a spanning tree of depth at most $\eta + 1$.

Each parcel P corresponds to a subgraph of G, namely the subgraph consisting of the bricks corresponding to the faces making up P. Let us refer to this subgraph as the *filled-in* version of P.

Step 3: Select a set of "artificial" terminals on the boundaries of parcels to achieve the following:

- for each filled-in parcel, there is a solution that is feasible with respect to original and artificial terminals whose cost is at most that of the parcel's boundary plus the cost of the intersection of OPT with the filled-in parcel, and
- the union over all parcels of such feasible solutions is a feasible solution for the original graph.

Step 4: Designate portals on the boundary of each brick.

Step 5: For each filled-in parcel P, find a optimal solution in the portal-connected graph, $\mathcal{B}^+(P)$. Output the union of these solutions.

Step 1 can be carried out in $O(n \log n)$ time [9]. Step 2 can be done in linear time via breadth-first search in the planar dual of MG, and then applying a "shifting" technique in the tradition of Baker [1]. Step 3 uses the fact that each parcel's boundary consists of edge-disjoint, noncrossing cycles. If such a cycle separates terminals, a vertex v on the cycle is designated an artificial terminal. We set $r(v) = 2$ if the cycle separates terminals with requirement 2 and $r(v) = 1$ otherwise. Under this condition, any feasible solution for the original graph must cross the cycle; by adding the edges of the cycle, we get a feasible solution that also spans the artificial terminal. Step 3 can be trivially implemented in linear time. Step 5 is achieved in linear time using dynamic programming (Section 6).

5 Proof of the Structure Theorem

We are now ready to prove the Structure Theorem for $\{0, 1, 2\}$-edge connectivity, Theorem 4.6. In order to formalize the notion of connectivity across the boundary of a brick, we use the following definition:

Definition 5.1 (Joining vertex)**.** *Let H be a subgraph of G and P be a subpath of ∂G. A* joining vertex *of H with P is a vertex of P that is the endpoint of an edge of $H - P$.*

We will use the following structural lemmas in simplifying OPT. The first two were used in proving a Structure Theorem for the Steiner tree PTAS [9]; in these, T is a tree and P is an ϵ-short path on the boundary of the graph in which T and P are embedded. The third is in fact a generalization of the second lemma that we require for maintaining two connectivity.

Lemma 5.2 (Simplifying a tree with one root, Lemma 10.4 [9])**.** *Let r be a vertex of T. There is another tree $\widehat{T}$ that spans r and the vertices of $T \cap P$ such that $\mathbf{c}(\widehat{T}) \leq (1 + 4 \cdot \epsilon)\mathbf{c}(T)$ and $\widehat{T}$ has at most $11 \cdot \epsilon^{-1.45}$ joining vertices with P.*

Lemma 5.3 (Simplifying a tree with two roots, Lemma 10.6 [9])**.** *Let p and q be two vertices of T. There is another tree $\widehat{T}$ that spans p and q and the vertices of $T \cap P$ such that $\mathbf{c}(\widehat{T}) \leq (1 + c_1\epsilon)\mathbf{c}(T)$ and $\widehat{T}$ has at most $c_2 \cdot \epsilon^{-2.5}$ joining vertices with P, where c_1 and c_2 are constants.*

Lemma 5.4. *Let $\mathcal{F}$ be a set of non-crossing trees whose leaves are vertices of ϵ-short boundary paths P and Q and such that each tree in the forest has leaves on both these paths. There is a cycle or empty set $\widehat{C}$, a set $\widehat{\mathcal{F}}$ of trees, and a mapping $\phi : \mathcal{F} \longrightarrow \widehat{\mathcal{F}} \cup \{\widehat{C}\}$ with the following properties*

- *For every tree T in $\mathcal{F}$, $\phi(T)$ spans T's leaves.*
- *For two trees T_1 and T_2 in F, if $\phi(T_i) \neq \widehat{C}$ for at least one of $i = 1, 2$ then $\phi(T_1)$ and $\phi(T_2)$ are edge-disjoint (taking into account edge multiplicities).*
- *The subgraph $\bigcup \widehat{\mathcal{F}} \cup \{\widehat{C}\}$ has $o(\epsilon^{-2.5})$ joining vertices with $P \cup Q$.*
- $\boldsymbol{c}(\widehat{C}) + \sum\{\boldsymbol{c}(T) \ : \ T \in \widehat{\mathcal{F}}\} \leq 3\boldsymbol{c}(Q) + (1 + d \cdot \epsilon)\sum\{\boldsymbol{c}(T) \ : T \in \mathcal{F}\}$ *where d is an absolute constant.*

Proof. View the embedding of the boundary such that P is on top and Q is at the bottom. Let $T_1, \ldots, T_k$ be the trees of F ordered according the order of their leaves from left to right.

There are two cases.

Case 1) $k > 1/\epsilon$. In this case, we reduce the number of trees by incorporating a cycle $\widehat{C}$. Let a be the smallest index such that $\boldsymbol{c}(T_a) \leq \epsilon \boldsymbol{c}(F)$ and let b be the largest index such that $\boldsymbol{c}(T_b) \leq \epsilon \boldsymbol{c}(F)$. We will replace trees $T_a, T_{a+1}, \ldots, T_b$ with a cycle. Let Q' be the minimal subpath of Q that spans the leaves of $\bigcup_{i=a}^{b} T_i$ on Q. We likewise define P'. Let L be the leftmost Q-to-P path in T_a and let R be the rightmost Q-to-P path in T_b. Since P is ϵ-short,

$$\boldsymbol{c}(P') \leq (1 + \epsilon)\boldsymbol{c}(L \cup Q' \cup R). \tag{5}$$

To obtain $\widehat{F}$ from F, we replace the trees $T_a, \ldots, T_b$ with the cycle $\widehat{C} = P' \cup L \cup Q' \cup R$ and set $\phi(T_a), \ldots, \phi(T_b)$ to $\widehat{C}$. By construction $\widehat{C}$ spans the leaves of $\cup_{i=a}^{b} T_i$.

Case 2) $k \leq 1/\epsilon$. In this case, the number of trees is already bounded. We set $a = 2, b = 1$ so as to not eliminate any trees, and we set $\widehat{C}$ to be the empty set.

In both cases, for each remaining tree T_i ($i \neq a, a+1, \ldots, b$) we do the following. Let T_i' be a minimal subtree of T_i that spans all the leaves of T_i on P and exactly one vertex r of Q. Let Q_i be the minimal subpath of Q that spans the leaves of T_i on Q. We replace T_i with the tree $\widehat{T_i}$ that is the union of Q_i' and the tree guaranteed by Lemma 5.2 for tree T_i' with root r and ϵ-short path P. By construction $\widehat{T_i}$ spans the leaves of T_i. We set $\phi(T_i) = \widehat{T_i}$ for $i \neq a, \ldots, b$.

$\widehat{C}$ has at most four joining vertices with $P \cup Q$. Each tree $\widehat{T_i}$ has one joining vertex with Q and, by Lemma 5.2, $o(\epsilon^{-1.5})$ joining vertices with P. By the choice of a and b, there are at most $2/\epsilon$ of the trees in the second part of the construction. This yields the bound on joining vertices.

The total cost of the replacement cycle is:

$$\begin{aligned}
\boldsymbol{c}(\widehat{C}) &\leq \boldsymbol{c}(P') + \boldsymbol{c}(L) + \boldsymbol{c}(Q') + \boldsymbol{c}(R) \\
&\leq (2+\epsilon)(\boldsymbol{c}(L) + \boldsymbol{c}(Q') + \boldsymbol{c}(R)) \qquad \text{by Equation (5)} \\
&\leq (2+\epsilon)(\boldsymbol{c}(T_a) + \boldsymbol{c}(Q') + \boldsymbol{c}(T_b)) \qquad \text{since } L \text{ and } R \text{ are paths in } T_a \text{ and } T_b \\
&\leq (2+\epsilon)(2\epsilon\boldsymbol{c}(F) + \boldsymbol{c}(Q')) \qquad \text{by the choice of } a \text{ and } b \\
&\leq (4\epsilon + 2\epsilon^2)\boldsymbol{c}(F) + (2+\epsilon)\boldsymbol{c}(Q')
\end{aligned}$$

The total cost of the replacement trees is:

$$\begin{aligned}
\sum_{i=1,\ldots,a-1,b+1\ldots k} \boldsymbol{c}(\widehat{T}_i) &\le \sum_{i=1,\ldots,a-1,b+1\ldots k} \boldsymbol{c}(Q'_i) + (1+4\epsilon)\boldsymbol{c}(T'_i) && \text{by Lemma 5.2} \\
&\le \sum_{i=1,\ldots,a-1,b+1\ldots k} \boldsymbol{c}(Q'_i) + (1+4\epsilon)\boldsymbol{c}(T_i) && \text{since } T'_i \text{ is a subtree of } T_i
\end{aligned}$$

By the ordering of the trees and the fact that they are non-crossing, Q' and the Q'_i's are disjoint. Combining the above gives the bound on cost. □

5.1 Construction of a new solution

We start with a brief overview of the steps used to prove the structure theorem. We start with an edge multiset forming an optimal solution, OPT. Each step modifies either the input graph G or a subgraph thereof while simultaneously modifies the solution. The graphs and edge multisets resulting from these steps are denoted by subscripts. Details are given in subsequent sections.

Augment We add two copies of each supercolumn, obtaining G_A and OPT_A. We consider the two copies to be interior to the two adjacent bricks. This step allows us, in the *restructure* step, to concern ourselves only with connectivity between the north and south boundaries of a brick.

Cleave *Cleaving* a vertex refers to splitting it into two vertices and adding an artificial edge between the two vertices. In the *cleave* step, we modify G_A (and so in turn modify MG_A and OPT_A) to create G_C (and MG_C and OPT_C) by cleaving certain vertices while maintaining a planar embedding. Let J_C be the set of artificial edges introduced. Note that $X_C/J_C = X_A$. The artificial edges are assigned zero cost so the metric between vertices is preserved. The artificial edges are added to the solution, possibly in multiplicity, so connectivity is preserved.

Flatten In this step, for each brick B, we consider the intersection of the solution with $\mathrm{int}(B)$; we replace some of the connected components of the intersection with subpaths of the boundary of B. We denote the resulting solution by OPT_F.

Map We map the edges of OPT_F to $\mathcal{B}^+(MG)$ creating OPT_M. This step temporarily disconnects the solution.

Restructure In this step, we modify the solution OPT_M. For each brick B in MG_C, the part of OPT_M strictly interior to B is replaced with another subgraph that has few joining vertices with ∂B. We denote the resulting solution by OPT_S.

Rejoin In order to re-establish connections broken in the *Map* step, we add edges to OPT_S. Next, we contract the artificial edges added in the *cleave* step. We denote the resulting solution by $\widehat{OPT}$.

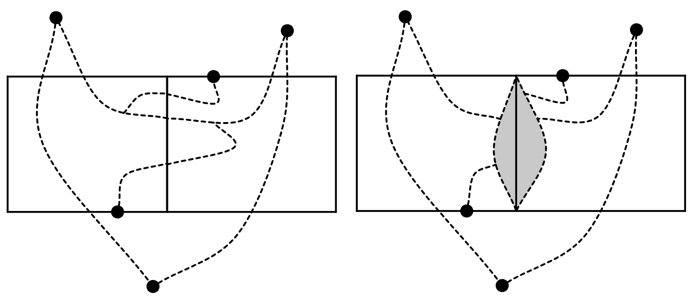
Note that the solutions OPT, OPT_A, and so on are multisets; an edge can occur more than once. We now describe these steps in greater detail.

5.1.1 Augment

Recall that a supercolumn is the eastern boundary of one brick and the western boundary of another, and that the sum of costs of all supercolumns is small. In the *Augment* step, for each supercolumn P, we modify the graph as shown in Figure 7:

- Add to the graph two copies of P, called P_1 and P_2, creating two new faces, one bounded by P_1 and P and the other bounded by P and P_2.
- Add P_1 and P_2 to OPT.

The resulting graph is denoted G_A, and the resulting solution is denoted OPT'_A. We consider P_1 and P_2 to be internal to the two bricks. Thus P remains part of the boundary of each of the bricks, and MG contains P but not P_1 or P_2. Since P_1 and P_2 share no internal vertices with P, the joining vertices of $\mathrm{OPT}'_A \cap B$ with ∂B belong to N_B and S_B.



(a) (b)

Figure 7: Adding the column between two adjacent bricks (solid) in the *augment* step. The dotted edges represent OPT in (a) and OPT_A in (b).

We perform one more step, a *minimality-achieving step*:

- We remove edges from OPT'_A until it is a minimal set of edges achieving the desired connectivity between terminals.

Let OPT_A be the resulting set. We get:

Lemma 5.5. *For every brick B, the joining vertices of* $\mathrm{OPT}_A \cap B$ *with ∂B belong to N_B and S_B.*

5.1.2 Cleave

We define a graph operation, *cleave*. Given a vertex v and a bipartition A, B of the edges incident to v, v is cleaved by

- splitting v into two vertices, v_A and v_B,
- mapping the endpoint v of edges in A to v_A,
- mapping the endpoint v of edges in B to v_b, and
- introducing a zero-cost edge $e_v = v_A v_B$.

This operation is illustrated in Figure 8(a) and (b). If the bipartition A, B is non-interleaving with respect to the embedding's cycle of edges around v then the construction maintains a planar embedding.

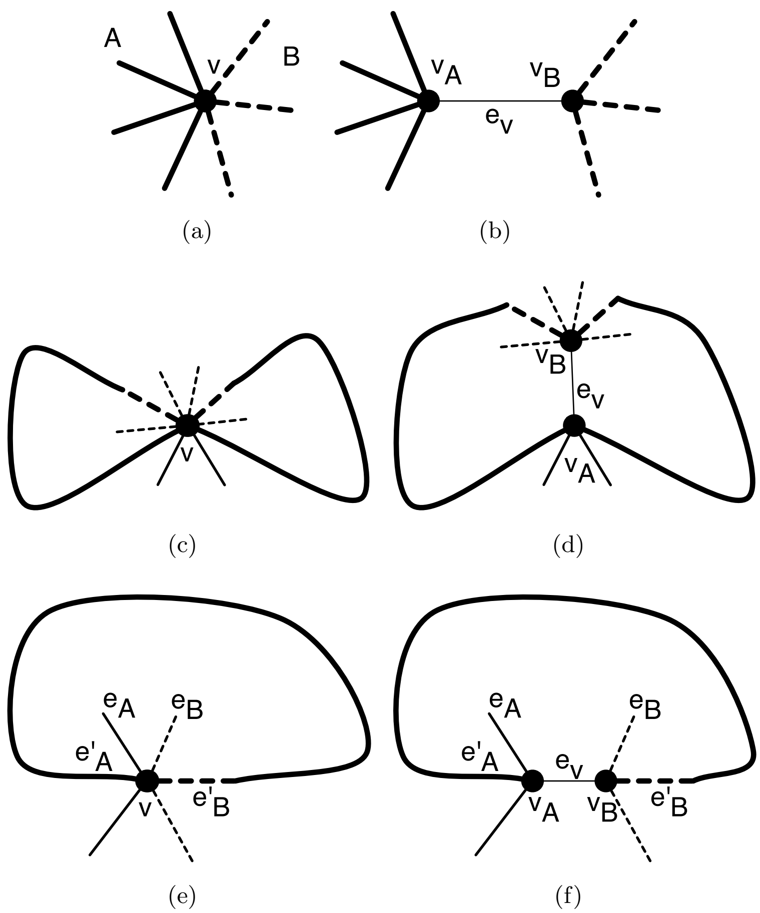


Figure 8: Cleavings illustrated. The bipartition of the edges incident to v is given by the dashed edges A and solid edges B. (a) Starting with this bipartition, (b) the result of cleaving vertex v according to this bipartition. A simplifying cleaving of vertex v with respect to a cycle (bold) before (c) and after (d). A lengthening cleaving of a cycle (e) before and (f) after.

We use two types of cleavings:

Simplifying cleavings Refer to Figures 8(c) and (d). Let C be a clockwise non-self-crossing, non-simple cycle that visits vertex v twice. Define a bipartition A, B of the edges incident to v as follows: given the clockwise embedding of the edges incident to v, let A start and end with consecutive edges of C and contain only two edges of C. Such a bipartition exists because C is non-self-crossing.

Lengthening cleavings Refer to Figures 8(e) and (f). Let C be a cycle, let v be a vertex on C with two edges e_A and e_B adjacent to v embedded strictly inside C, and let e'_A and e'_B be consecutive edges of C adjacent to v such that the following bipartition is non-crossing

with respect to the embedding: A, B is a bipartition of the edges adjacent to v such that $e_A, e'_A \in A$ and $e_B, e'_B \in B$.

We perform simplifying cleavings for non-simple cycles of OPT_A until every cycle is simple; the artificial edges introduced are not included in OPT. The following lemma does not use planarity and shows that (since cycles get mapped to cycles in this type of cleaving) simplifying cleavings preserve two-edge connectivity.

Lemma 5.6. *Let e be an edge in a graph H. Let $\widehat{H}$ be the graph obtained from H by a simplifying cleaving. Then e is a cut-edge in H iff it is a cut-edge in $\widehat{H}$.*

Proof. Let u_1, u_2 be the endpoints of e and let C be the cycle w.r.t. which a simplifying cleaving was performed. If H contains an e-avoiding u_i-to-C path for $i = 1, 2$ then e is not a cut-edge in H, and similarly for $\widehat{H}$. Suppose therefore that removing e separates u_i from C in H. Then the same is true in $\widehat{H}$, and conversely. $\square$

Corollary 5.7. *For $k = 1, 2$, if two vertices are k-edge connected in H then any of their copies are k-edge connected in H_C.*

Moreover, after all the simplifying cleavings, every cycle is simple, so:

Lemma 5.8. *Vertices that are two-edge-connected in* OPT_C *are biconnected.*

Next we perform lengthening cleavings w.r.t. the boundary of a brick and edges e_A and e_B of OPT_C; we include in OPT_C all the artificial zero-cost edges introduced. Lengthening cleavings clearly maintain connectivity. Suppose that vertices x and y are biconnected in OPT_C, and consider performing a lengthening cleaving on a vertex v. Since there are two internally vertex-disjoint x-to-y paths in OPT_C, v cannot appear on both of them. It follows that there remain two internally vertex-disjoint x-to-y paths after the cleaving. We obtain the following lemma.

Lemma 5.9. *Lengthening cleavings maintain biconnectivity.*

Lengthening cleavings are performed while there are still multiple edges of the solution embedded in a brick that are incident to a common boundary vertex. Let J_C be the set of artificial edges that are introduced by simplifying and lengthening cleavings. We denote the resulting graph by G_C, we denote the resulting mortar graph by MG_C, and we denote the resulting solution by OPT_C.

As a result of the cleavings, we get the following:

Lemma 5.10. *Let B be a brick in G_C with respect to MG_C. The intersection* $\mathrm{OPT}_C \cap int(B)$ *is a forest whose joining vertices with ∂B are the leaves of the forest.*

Proof. Let H be a connected component of $\mathrm{OPT}_C \cap \mathrm{int}(B)$. As a result of the lengthening cleavings, the joining vertices of H with ∂B have degree 1 in H. Suppose otherwise; then there is a vertex v of $H \cap \partial B$ that has degree > 1 in H. Hence v is a candidate for a lengthening cleaving, a contradiction.

By Theorem 2.5 and the minimality-achieving step of the Augment step, any cycle in H must include a terminal u with $r(u) = 2$ by Theorem 2.5. Since there are no terminals strictly enclosed by bricks, u must be a vertex of ∂B. However, that would make u a joining vertex of H with ∂B.

As argued above, such vertices are leaves of H, a contradiction to the fact that u is a vertex of a cycle in H. Therefore H is acyclic.

Furthermore, leaves of H are vertices of ∂B since OPT_C is minimal with respect to edge inclusion and terminals are not strictly internal to bricks. □

Lemma 5.11. *Let C be a cycle in OPT_C. Let B be a brick. Distinct connected components of $C \cap int(B)$ belong to distinct components of $\mathrm{OPT}_C \cap int(B)$.*

Proof. Assume the lemma does not hold. Then there is a C-to-C path P in $\mathrm{int}(B)$. Each vertex of P that is strictly interior to B is not a terminal. A vertex of P that was on $\partial(B)$ would be a candidate for a lengthening cleaving, a contradiction. Therefore P includes no terminals. This contradicts Theorem 2.8. □

5.1.3 Flatten

For each brick B, consider the edges of OPT_C that are strictly interior to B. By Lemma 5.10, the connected components are trees. By Lemma 5.5, for each such tree T, every leaf is either on B's northern boundary N_B or on B's southern boundary S_B. For each such tree T whose leaves are purely in N_B, replace T with the minimal subpath of N_B that contains all the leaves of T. Similarly, for each such tree T whose leaves are purely in S_B, replace T with the minimal subpath of S_B that contains all the leaves of T.

Let OPT_F be the resulting solution. Note that OPT_F is a multiset. An edge of the mortar graph can appear with multiplicity greater than one.

5.1.4 Map

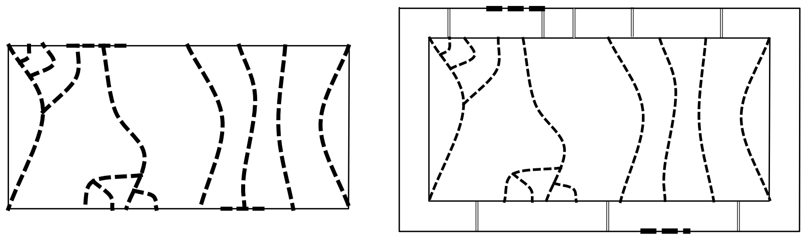
This step is illustrated in Figures 9(a) and (b). In this step, the multiset OPT_F of edges resulting from the flatten step is used to select a set OPT_M of edges of $\mathcal{B}^+(MG_C)$. Recall that every edge e of MG_C corresponds in $\mathcal{B}^+(MG_C)$ to three edges: two brick copies (one in each of two bricks) and one mortar-graph copy. In this step, for every edge e of MG_C, we include the mortar-graph copy of e in OPT_M with multiplicity equal to the multiplicity of e in OPT_F. At this point, none of the brick copies are represented in OPT_M.

Next, recall that in the augment step, for each supercolumn P, we created two new paths, P_1 and P_2, and added them to OPT. The edges of these two paths were not considered part of the mortar graph, so mortar-graph copies were not included in OPT_M for these edges. Instead, for each such edge e, we include the brick-copy of e in OPT_F with multiplicity equal to the multiplicity of e in OPT_F.

Finaly, for each edge e interior to a brick, we include e in OPT_M with the same multiplicity as it has in OPT_F.

5.1.5 Restructure

Let B be a brick. For simplicity, we write the boundary paths of B as N, E, S, W. Let F be the multiset of edges of OPT_M that are in the interior of B. F is a forest (Lemma 5.10). As a result of the *flatten* step, each component of F connects S to N. We will replace F with another subgraph $\widehat{F}$ and map each component T of F to a subgraph $\phi(T)$ of $\widehat{F}$ where $\phi(T)$ spans the leaves of T and



(a) (b)

Figure 9: Map (a) The intersection of OPT_C with a brick, dashed. (b) The same brick in the portal connected graph with portal edges (double-lines) connecting the brick to the corresponding face (outer boundary) of the mortar graph.

is a tree or a cycle. Distinct components of F are mapped by ϕ to edge-disjoint subgraphs (taking into account multiplicities).

Refer to Figure 10. We inductively define S-to-N paths $P_0, P_1, \ldots$ and corresponding integers $k_0, k_1, \ldots$. Let $s_0, \ldots, s_t$ be the vertices of S guaranteed by Lemma 4.2 (where s_0 is the vertex common to S and W and s_t is the vertex common to S and E). Let P_0 be the easternmost path in F from S to N. Let k_0 be the integer such that $\mathrm{start}(P_0))$ is in $S[s_{k_0}, s_{k_0+1})$. Inductively, for $i \geq 0$, let P_{i+1} be the easternmost path in $F_{N \wedge S}$ from $S[s_0, s_{k_i})$ to N that is vertex-disjoint from P_i. Let k_i be the integer such that $\mathrm{start}(P_i) \in S[s_{k_i}, s_{k_i+1})$. This completes the inductive definition of $P_0, P_1, \ldots$. Note that the number of paths is at most t, which in turn is at most $\kappa(\epsilon)$ as defined in Equation 2.

We use these paths to decompose F, as illustrated in Figure 10. Let F_i be the set of edges of $F - P_{i+1}$ enclosed by the cycle formed by P_i, P_{i+1}, N and S. Clearly $F = \cup_i F_i$. If P_i is connected to P_{i+1}, they share at most one vertex, w_i. If they are not connected, we say w_i is undefined.

There are two cases: either F_i is connected or not.

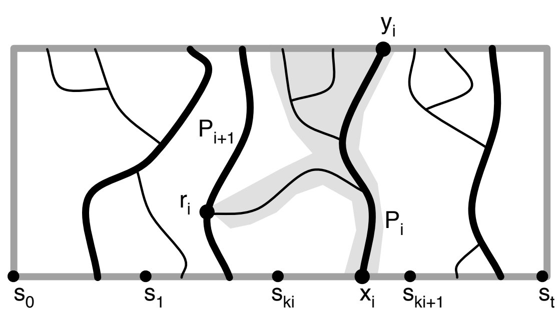


Figure 10: Paths used to decompose F. The brick boundary is given by the rectangle. The paths $P_0, P_1, \ldots$ are bold. F_i is given by the shaded background.

Connected case: There are two subcases. Either F_i spans vertices of $S[\cdot, s_{k_i})$ or not.

Suppose F_i spans vertices of $S[\cdot, s_{k_i})$. Let T_S be a minimal subtree of F_i that spans $F_i \cap S$ and let T_N be a minimal subtree of F_i that spans $F_i \cap N$. Let r_N be the first vertex of P_i in T_N and let

r_S be the last vertex of P_i in T_S. A path in F_i from T_S to N that does not go through r_S contradicts the choice of P_{i+1} as there would be a path in F_i from $S[\cdot, s_{k_i})$ to N that is disjoint from P_i. It follows that T_S and T_N are edge disjoint: if they intersect they may only do so at $r_S = r_N$. If w_i is defined, then there is a path Q from w_i to P_i; Q intersects P_i between r_N and r_S, for otherwise there would be a superpath of Q that contradicts the choice of P_{i+1}.

If w_{i-1} is defined and $w_{i-1} \in T_N$, then we replace T_N with the tree guaranteed by Lemma 5.3 with roots r_N and w_{i-1}. Otherwise, we replace T_N with the tree guaranteed by Lemma 5.2 with root r_N. We do the same for T_S.

Suppose F_i does not span vertices of $S[\cdot, s_{k_i})$. Let T_N be a minimal connected subgraph of $F_i \cup S[s_{k_i}, \text{start}(P_i)]$ that spans $F_i \cap N$. Let r_N be the first vertex of P_i in T_N. If w_i is defined, then there is a path Q from w_i to $P_i \cup S[s_{k_i}, \text{start}(P_i)]$ and Q's intersection with P_i belongs to $P_i[\cdot, r_N]$, for otherwise there would be a superpath of Q that contradicts the choice of P_{i+1}. If w_{i-1} is defined and $w_{i-1} \in T_N$, then we replace F_i with the tree guaranteed by Lemma 5.3 with roots r_N and w_{i-1} along with Q, $P_i[\cdot, r_N]$, and $S[s_{k_i}, \text{start}(P_i)]$. Otherwise we replace F_i with the tree guaranteed by Lemma 5.2 with root r_N along with Q, $P_i[\cdot, r_N]$, and $S[s_{k_i}, \text{start}(P_i)]$.

In both cases, we define $\phi'(F_i)$ to be the resulting tree that replaces F_i. By construction, $\phi'(F_i)$ spans the leaves of F_i and w_{i-1} and w_i (if defined).

Disconnected case: In this case, by the definition of P_{i+1}, $F_i \cap S$ is a subset of the vertices of $S[s_{k_i}, \text{start}(P_i)]$, for otherwise there would be a path to the right of P_{i+1} that connects to N and is disjoint from P_i.

If F_i is connected to F_{i+1}, then the western-most tree T_W is a tree with root w_i and leaves on S and does not connect to N as that would contradict the choice of P_{i+1}; if this is the case, let $\widehat{T}_W$ be the tree guaranteed by Lemma 5.2 and define $\phi'(T_W) = \widehat{T}_W$.

If F_i is connected to F_{i-1}, let S' be the subpath of S that spans the eastern-most tree T_E's leaves on S. Let $\widehat{T}_E$ be the tree guaranteed by Lemma 5.3 that spans the eastern-most tree's leaves on N and roots w_{i-1} and $\text{start}(P_i)$ and define $\phi'(T_E) = \widehat{T}_E$.

Let $\mathcal{F}$ be the set of remaining trees, let $P = N$, and let $Q = S[s_{k_i}, \text{start}(P_i)]$ in Lemma 5.4. Let $\widehat{C}$, $\widehat{\mathcal{F}}$, and ϕ be the cycle (or empty set), set of trees, and mapping that satisfy the properties stated in the lemma.

We define $\widehat{F}_i$ to consist of the trees of $\widehat{\mathcal{F}}$ and the cycle $\widehat{C}$ (and $\widehat{T}_W$, S' and $\widehat{T}_E$ if defined).

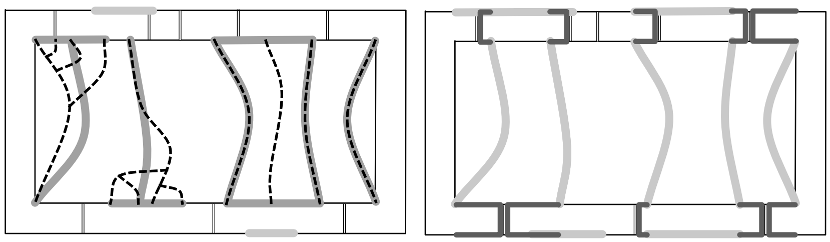
We replace every F_i with $\widehat{F}_i$, as described above, for every brick, creating OPT_S. This is illustrated in Figure 11(a). Now we define ϕ in terms of ϕ'. A component T of F is partitioned into adjacent trees in this restructuring: namely $T_1, \ldots, T_k$, $k \geq 1$. T_1 and T_k may be restructured via the disconnected case and all others are restructured via the connected case. Define $\phi(T) = \cup_{i=1}^{k} \phi'(T_i)$. If $k > 1$, then consecutive trees T_i and T_{i+1} share a vertex w_i and by construction $\phi'(T_i)$ and $\phi'(T_i)$ also share this vertex. Since $\phi'(T)$ spans the leaves of T, we get that $\phi(T)$ spans the leaves of T, as desired. Also by construction, the submapping of ϕ' of trees to trees (and not cycles) is bijective; the same holds for ϕ.

Number of joining vertices In both the connected and disconnected case, the number of leaves is the result of a constant number of trees resulting from Lemmas 5.2, 5.3 and 5.4. Therefore, $\widehat{F}_i$ has $o(\epsilon^{-2.5})$ joining vertices with N and S. Since $i \leq \kappa(\epsilon) = O(\epsilon^{-3})$, OPT_S has $o(\epsilon^{-5.5})$ joining

vertices with the boundary of each brick. This is the number of connections required to allow a solution to be nearly optimal and affects the number of portals required in Equation (3):

$$\alpha(\epsilon) \text{ is } o(\epsilon^{-5.5}) \tag{6}$$

This will allow us to prove the second part of the Structure Theorem.



(a) (b)

Figure 11: Continuing from Figure 9, restructure and rejoin: (a) Restructured version (dark grey) of the intersection of OPT_M with the brick (dashed). (b) Connecting the restructured solution inside the brick to the mortar graph through portals (via dark grey edges).

5.1.6 Rejoin

In this step, we make inter-brick connections for parts that were disconnected in the *mapping* step. Since joining vertices represent the ends of all disconnected parts, it suffices to connect joining vertices of OPT_S with ∂B to their mortar-graph counterparts via portal edges.

This is illustrated in Figure 11 (d): We first move the edges of $\mathrm{OPT}_S \cap \partial B$ to MG for every brick B. Such edges may have been introduced in the restructure step: for every brick B, we connect $\mathrm{OPT}_S \cap B$ to the mortar graph. For every joining vertex v of $\mathrm{OPT}_S \cap B$, we find the nearest portal p_v, add the subpath of ∂B and MG connecting v and p_v and add the portal edge corresponding to p_v. Finally we contract the edges introduced in the cleaving step. This produces a solution $\widehat{OPT}$ of $\mathcal{B}^+(MG)$.

5.2 Analysis of connectivity

In the augment step, because the added paths P_1 and P_2 form a cycle, this transformation preserves two-connectivity and connectivity between terminals. Cleaving clearly preserves connectivity and, by Lemmas 5.8 and 5.9, terminals that require two-connectivity are biconnected in OPT_C. Therefore, for terminals x and y requiring connectivity, there is a path P_C in OPT_C connecting them. If x and y require two-connectivity, there is a simple cycle C_C in OPT_C connecting them. We follow P_C and C_C through the remaining steps.

Flatten Consider a tree T that is replaced by a subpath Q of a northern or southern boundary of a brick that spans T's leaves. Q spans any terminals that T spanned, since there are no terminals internal to bricks.

$P_C \cap T$ is therefore a (set of) leaf-to-leaf paths and so $(P_C - T) \cup Q$ contains an x-to-y path. It follows that there is an x-to-y path P_F in OPT_F.

By Lemma 5.11, $C_C \cap T$ is a single path and, by the above reasoning, is a leaf-to-leaf path. Therefore $(C_C - T) \cup Q$ contains a cycle through x and y. It follows that there is a cycle C_F through x and y in OPT_F

Map P_F (C_F) gets mapped to a sequence $\mathcal{P}_M = (P_M^1, P_M^2, \ldots)$, (a cyclic sequence $\mathcal{C}_M = (C_M^1, C_M^2, \ldots)$) of paths of OPT_M in $\mathcal{B}^+(MG_C)$ such that each path either consists completely of mortar-graph edges or consists completely of brick-copy edges. The last vertex of one path and the first vertex of the next path are copies of the same vertex of G_C, and that vertex belongs to a north or south boundary of MG_C. By Lemma 5.10, each path in $\mathcal{P}_M$ or $\mathcal{C}_M$ that consists of brick-copy edges starts and ends at the northern or southern boundary of a brick.

Restructure We define a mapping $\hat{\phi}$, based in part on the map ϕ defined in Section 5.1.5.. For a path Q in $\mathcal{P}_M$ or $\mathcal{C}_M$ that uses mortar-copy edges, define $\hat{\phi}(Q) = Q$. For a path Q in $\mathcal{P}_M$ or $\mathcal{C}_M$ that uses brick-copy edges, let T be the tree in OPT_M that contains Q and define $\hat{\phi}(Q) = \phi(T)$.

Let $\mathcal{C}_S$ be the cyclic sequence of trees and cycles to which $\mathcal{C}_M$ maps by ϕ. Since $\phi(T)$ spans the leaves (joining vertices) of T, consecutive trees/cycles in $\mathcal{C}_S$ contain copies of the same vertex. By Lemma 5.11 and that, within a brick, the preimage of the set of trees mapped to by ϕ, the trees of $\mathcal{C}_S$ are edge disjoint. (The cycles may be repeated.)

Likewise we define $\mathcal{P}_S$ having the same properties except for the fact that the sequence is not cyclic.

Rejoin This step reconnects $\mathcal{P}_S$ and $\mathcal{C}_S$.

Consider a tree T in either of these sequences that contains brick-copy edges. The rejoin step first moves any edge of T that is in a brick boundary to the mortar copy. It then connects joining vertices to the mortar by way of detours to portals and portal edges. Therefore, T is mapped to a tree T_J that connects the mortar copies of T's leaves.

Consider a cycle C in either $\mathcal{P}_S$ or $\mathcal{C}_S$. C contains a subpath of N and S whose edges are moved to their mortar copy and two N-to-S paths whose joining vertices are connected to their mortar copies. Therefore C is mapped to a cycle C_J through the mortar copies of the boundary vertices of C.

Let the resulting sequence and cyclic sequence of trees and cycles be $\mathcal{P}_J$ and $\mathcal{C}_J$. Since consecutive trees/cycles in $\mathcal{C}_S$ contain copies of the same vertex, in $\mathcal{C}_J$ consecutive trees/cycles contain common mortar vertices. We have that $\mathcal{C}_J$ is a cyclic sequence of trees and cycles, through x and y, the trees of which are edge-disjoint. Therefore the union of these contains a cycle through x and y.

Similarly, we argue that the union of the trees and cycles in $\mathcal{P}_J$ contains and x-to-y path.

5.3 Analysis of cost increase

By Lemma 4.3, the total costs of all the east and west boundaries of the bricks is an ϵ fraction of OPT, so we have

$$\boldsymbol{c}(\mathrm{OPT}_A) \leq (1 + 2\epsilon)\boldsymbol{c}(\mathrm{OPT}). \tag{7}$$

The cleaving step only introduces edges of zero cost, so

$$\boldsymbol{c}(\mathrm{OPT}_C) = \boldsymbol{c}(\mathrm{OPT}_A). \tag{8}$$

The flatten step replaces trees by ϵ-short paths, and so can only increase the cost by an ϵ fraction, giving:

$$\boldsymbol{c}(\mathrm{OPT}_F) \leq (1+\epsilon)\boldsymbol{c}(\mathrm{OPT}_C). \tag{9}$$

The mapping step does not introduce any new edges, so

$$\boldsymbol{c}(\mathrm{OPT}_M) = \boldsymbol{c}(\mathrm{OPT}_F). \tag{10}$$

The restructure step involves replacing disjoint parts of $\mathrm{OPT}_M \cap B$ for each brick by applying Lemmas 5.2, 5.3, and 5.4. This increases the cost of the solution by at most an $O(\epsilon)$ fraction. Further we add subpaths of $S[s_{k_i}, \mathrm{start}(P_i)]$ where OPT_M contains disjoint subpaths P_i from $\mathrm{start}(P_i)$ to N and by the brick properties, $\boldsymbol{c}(S[s_{k_i}, \mathrm{start}(P_i)]) \leq \boldsymbol{c}(P_i)$. This increases the the cost of the solution by at most another $O(\epsilon)$ fraction. We get, for some constant c:

$$\boldsymbol{c}(\mathrm{OPT}_S) \leq (1+c\epsilon)\boldsymbol{c}(\mathrm{OPT}_M). \tag{11}$$

In the rejoin step, we add two paths connecting a joining vertex to its nearest portal: along the mortar graph and along the boundary of the brick. The cost added per joining vertex is at most twice the interportal cost: at most $2\boldsymbol{c}(\partial B)/\theta$ for a joining vertex with the boundary of brick B, by Lemma 4.4. Since each mortar graph edge appears as a brick-boundary edge at most twice, the sum of the costs of the boundaries of the bricks is at most $18\epsilon^{-1}\mathrm{OPT}$ (Equation (1)). Since there are $\alpha(\epsilon)$ joining vertices of OPT_S with each brick, the total cost added due to portal connections is at most $36\frac{\alpha}{\theta\epsilon}\mathrm{OPT}$. Replacing α and θ via Equations (6) and (3) gives us that the total cost added due to portal connections is

$$O(\epsilon\boldsymbol{c}(\mathrm{OPT})) \tag{12}$$

Combining equations (7) through (12), $\boldsymbol{c}(\widehat{\mathrm{OPT}}) \leq (1+c'\epsilon)\boldsymbol{c}(\mathrm{OPT})$ for some constant c', proving Theorem 4.6.

6 Dynamic Program

In this section we give the details of finding an optimal solution to the survivable network design problem in the portal-connected graph $\mathcal{B}^+(P)$ for a parcel P. Recall that $\mathcal{B}^\div(B)$ is the brick contracted graph, in which each brick of $\mathcal{B}^+(P)$ is contracted to a vertex.

Recall that a parcel P is a subgraph of MG and defines a set of bricks contained by the faces of P. The planar dual of the parcel has a spanning tree of depth $\eta+1$. Since each brick has at most $\theta+1$ portals, it follows that the planar dual of $\mathcal{B}^\div(P)$ has a spanning tree of depth at most $(\eta+1)(\theta+1)$. It follows that there is a rooted spanning tree of $\mathcal{B}^\div(P)$ (the primal) such that, for each vertex v, there at most $2(\eta+1)(\theta+1)+1$ edges from descendents of v to non-descendents. This spanning tree, $\widehat{T}$, rooted at the infinite face, is used to guide the dynamic program.

6.1 The dynamic programming table

Here we relate the tree $\widehat{T}$, which spans the brick-contracted parcel, to the portal-connected parcel. Let $\widehat{T}(v)$ denote the subtree of $\widehat{T}$ rooted at v. For each vertex v of $\widehat{T}$, define

$$f(v) = \begin{cases} B & \text{if } v \text{ is the result of contracting a brick } B \\ v & \text{otherwise} \end{cases}$$

and define $W(v)$ to be the subgraph of $\mathcal{B}^+(P)$ induced by $\bigcup\{f(w) : w \in \widehat{T}(v)\}$. Let $\delta(S)$ be the subset of edges with exactly one endpoint in the subgraph S (i.e. a cut). It follows that the cut $\delta(W(v))$ in $\mathcal{B}^+(P)$ is equal to the cut $\delta(\widehat{T}(v))$ in $\mathcal{B}^{\div}(P)$ and so has $O(\theta\eta)$ edges. The dynamic programming table will be indexed over relationships of these edges that we call configurations. Recall that we have doubled the edges in our graph, to allow for multiple use. Likewise, we double the portal edges.

6.1.1 Configurations

Let $L = \delta(H)$ for a subgraph H of G. We define a *configuration* K_L corresponding to L, illustrated in Figure 12. First, *cut the adjacencies* between edges of L that have a common endpoint in $G-H$, creating a new set of edges L': if two edges a and b of L have a common endpoint x in $G-H$, introduce a new vertex x' such that a's endpoint in $G-H$ is x and b's endpoint is x'. The edges in L are identified with the corresponding edge in L'. A configuration K_L is a forest with no degree-2 vertices whose leaf edges are a subset of L' (and so also L). Cutting the connectivity of L in $G-H$ implies that if $e \in K_L \cap L$ then e is a leaf edge of K_L. We denote the set of all configurations on edge set L by $\mathcal{K}_L$.

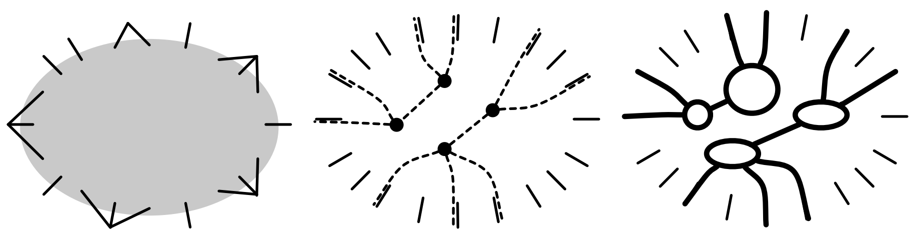


Figure 12: (a) The cut edges L (black) of a subgraph H (shaded). (b) A configuration (dotted forest) for a set of edges L' derived from L. (c) A subgraph (bold) that meets the configuration.

Lemma 6.1. *The number of configurations for a set of n edges is at most $2^n(2n)^{2n-2}$ and these trees can be computed in $O(2^n(2n)^{2n-2})$ time.*

Proof. Cayley's formula states that the number of trees with n vertices is n^{n-2}. Since a tree with no degree-two vertices at most n leaves can have at most $2n$ vertices, the number of such trees is at most $(2n)^{2n-2}$. It follows that the number of forests with n leaves and no degree two vertices is at most $2^n(2n)^{2n-2}$.

The set of trees can be computed in $O(1)$ amortized time per tree [23]. □

Connecting A configuration $K_L \in \mathcal{K}_L$ is *connecting* if

$$|K_L \cap L| \geq \begin{cases} 2 & \text{if } L \text{ separates terminals } x \text{ and } y \text{ such that } \boldsymbol{r}(x) = \boldsymbol{r}(y) = 2 \\ 1 & \text{if } L \text{ separates terminals.} \end{cases}$$

Compatibility Configurations $K_A \in \mathcal{K}_A$ and $K_B \in \mathcal{K}_B$ are *compatible* if for every edge $e \in A \cap B$ either $e \in K_A \cap K_B$ or $e \notin K_A \cup K_B$.

Consistency Given a graph G, let $EC(G)$ denote the graph whose vertices are the two-edge-connected components of G and whose edges denote adjacency of these components in G. This structure is familiarly known as a block-cut tree [12]. It is easy to see that $EC(G)$ is a forest. Let $\widetilde{EC}(G)$ be the graph obtained from $EC(G)$ by replacing every maximal path having internal degree 2 vertices with an edge. We say a configuration K_A is consistent with a set of mutually compatible configurations $\{K_{A_1}, K_{A_2}, \ldots\}$ if K_A is isomorphic to the graph $\widetilde{EC}(\cup_i K_{A_i})$.

Meeting Let H and M be subgraphs of G. Let M_H be the intersection of M with $H \cup \delta(H)$ after cutting the adjacencies of the edges in $G - H$. M *meets* a configuration $K_{\delta(H)}$ if $\widetilde{EC}(M_H) = K_{\delta(H)}$.

DP table entry We index the dynamic programming table DP_v for a subtree $\widehat{T}(v)$ of $\widehat{T}$ by the configurations $\mathcal{K}_{\delta(W(v))}$. The entry $DP_v[K]$ is the cost of the subgraph induced by the vertices $W(v)$ of a minimum-cost subgraph that *meets* configuration K. Note that the cost of no edge in $\delta(V(W(v)))$ is included in DP_v.

6.1.2 The filling procedure

If u is not a leaf of $\widehat{T}$, then we populate the entries of DP_u with the procedure FILL. We use the shorthand K for $K_{\delta(\{u\})}$ and K_i for $K_{\delta(W(u_i))}$. The cuts are with respect to the graph $\mathcal{B}^+(P)$.

FILL(DP_u)
Initialize each entry of DP_u to ∞.
Let $u_1, \ldots, u_s$ be the children of u.
For every set of connecting, mutually compatible configurations $K, K_1, \ldots, K_s$,
For every connecting configuration K_0 that is consistent with $K, K_1, \ldots, K_s$,
cost $\leftarrow \boldsymbol{c}(K \cap (\cup_{i=1}^s K_i)) + \boldsymbol{c}(\cap_{i=1}^s K_i) + \sum_{i=1}^s DP_{u_i}[K_i]$.
$DP_u[K_0] \leftarrow \min\{DP_u[K_0], \text{cost}\}$

If u is a leaf of $\widehat{T}$ and u does not correspond to a brick (i.e. $f(u) = u$), the problem is trivial. Each configuration K is a star: u is the center vertex, and the edges of $\delta(\{u\})$ are the edges of K. Since the cost of any subgraph that is induced by $\{u\}$ is zero, the value of $\textsc{Tab}_u[K]$ is zero for every K.

If u is a leaf of $\widehat{T}$ and $f(u) = B$, a brick. Let K be a configuration corresponding to the edges of the cut $\delta(W(v))$ in $\mathcal{B}^+(P)$. This is the set of portal edges corresponding to brick B. Let H_K be the minimum-cost subgraph of $B \cup \delta(W(v))$ that meets the configuration K. We know by Theorem 4.6 that H_K is the union of trees. Since there are at most θ portal edges corresponding to any brick, there are at most 2θ leaf edges in K, including duplicate edges. We can enumerate over all possible partitions of subsets of these leaf edges and compute the optimal Steiner tree connecting each set in the partition in $O(\theta^3 n)$ time (Theorem 2.1).

6.1.3 Running time

Consider the time required to populate the DP_u for all the leaves u of $\widehat{T}$. We need only consider non-crossing partitions of subsets of $\delta(W(v))$ since H_K is the union of non-crossing trees. The number of non-crossing partitions of an n element ordered set is the n^{th} Catalan number, which is at most $4^n/(n+1)$. Therefore, the number of non-crossing sub-partitions is at most 4^n. It follows

that the time to populate DP_v for v a brick vertex is $O(\theta^4 4^\theta |B|)$ which is $O(2^{\text{poly}(1/\epsilon)}|B|)$ since θ depends polynomially on $1/\epsilon$. Since a vertex appears at most twice in the set of bricks, the time needed to solve all the base cases in $O(2^{\text{poly}(1/\epsilon)}n)$ where n is the number of vertices in the parcel.

Consider the time required to populate the DP_u for all the internal vertices u of $\widehat{T}$. The number of edges in $\delta(W(v))$ in $\mathcal{B}^+(P)$ is $O(\theta\eta)$. By Lemma 6.1, it follows that the corresponding number of configurations is $O(2^{\text{poly}(1/\epsilon)})$ since θ and η each depend polynomially on $1/\epsilon$. There are $O(n)$ vertices of the recursion tree and so the time required for the dynamic program, not including the base cases is $O(2^{\text{poly}(1/\epsilon)}n)$.

The total running time of the dynamic program is $O(2^{\text{poly}(1/\epsilon)}n)$ where n is the number of vertices in the parcel.

6.2 Correctness

The connecting property guarantees that the final solution is feasible (satisfying the conectivity requirements). The definitions of compatible and consistent guarantee the inductive hypothesis.

We show that the procedure FILL correctly computes the cost of a minimum-cost subgraph H_u of $W(u)$ that meets the configuration K_0. We have shown that this is true for the leaves of the recursion tree. Since K is the configuration corresponding to the cut $\delta(\{u_0\})$, K is a star. Therefore $\boldsymbol{c}(K)$ is the cost of the edges of $\delta(\{u_0\})$: K is both the configuration and a minimum-cost subgraph that meets that configuration. Further, $\boldsymbol{c}(K \cap (\cup_{i=1}^s K_i))$ is the cost of the edges of K that are in K_i (for $i = 1, \ldots, s$). $w(\cap_{i=1}^s K_i)$ is equal to the cost of the edges common to K_1 and K_2 if $s = 2$ and zero otherwise. By the inductive hypothesis the cost computed is that of a H_u: the subgraph of $W(u)$ of a minimum-cost graph that meets this configuration.

Consider the entries of DP_r where r is the root of $\widehat{T}$. Since $\delta(W(r))$ is empty, there is only one configuration corresponding to this subproblem: the trivial configuration. Therefore, the dynamic program finds the optimal solution in $\mathcal{B}^+(P)$.

As argued in Section 4.4, combining parcel solutions forms a valid solution in our input graph. We need to compare the cost of the output to the cost of an optimal solution.

Recall that new terminals are added at the parcel boundaries to guarantee connectivity between the parcels; let $\boldsymbol{r}^+$ denote the requirements including these new terminals. Let $S(G, \boldsymbol{r})$ denote the optimal solution in graph G with requirements $\boldsymbol{r}$.

For each parcel P, there is a (possibly empty) solution S_P in $\mathcal{B}^+(P)$ for the original and new terminals in P consisting of edges of $S(\mathcal{B}^+(MG), \boldsymbol{r}) \cup \partial\mathcal{H}$ (where $\mathcal{H}$ is the set of parcels and $\partial\mathcal{H}$ is the set of boundary edges of all parcels). We have:

$$\boldsymbol{c}(S(\mathcal{B}^+(MG), \boldsymbol{r}) \cap \mathcal{B}^+(P)) \leq \boldsymbol{c}(S_P) = \boldsymbol{c}(S_P - \partial\mathcal{H}) + \boldsymbol{c}(S_P \cap \partial\mathcal{H}).$$

Every edge of $S(\mathcal{B}^+(MG), \boldsymbol{r})$ not in $\partial\mathcal{H}$ appears in S_P for exactly one parcel P, and so

$$\sum_{P \in \mathcal{H}} \boldsymbol{c}(S_P - \partial\mathcal{H}) \leq \boldsymbol{c}(S(\mathcal{B}^+(MG), \boldsymbol{r})).$$

Every edge of $\partial\mathcal{H}$ appears in at most two parcels, and so

$$\sum_{P \in \mathcal{H}} \boldsymbol{c}(S_P \cap \partial\mathcal{H}) \leq 2 \cdot \boldsymbol{c}(\partial\mathcal{H}).$$

Since a feasible solution for the original and new terminals in $\mathcal{B}^+(MG)$ can be obtained by adding a subset of the edges of $\partial\mathcal{H}$ to $S(\mathcal{B}^+(MG), \boldsymbol{r})$, the cost of the output of our algorithms is at most

$$\boldsymbol{c}(\partial\mathcal{H}) + \sum_{P\in\mathcal{H}} S(\mathcal{B}^+(P), \boldsymbol{r}^+) \leq \boldsymbol{c}(S(\mathcal{B}^+(MG), \boldsymbol{r})) + 3\boldsymbol{c}(\partial\mathcal{H}).$$

Combining the cost of the parcel boundaries, the definition of η, and the cost of the mortar graph, we obtain $\boldsymbol{c}(\partial\mathcal{H}) \leq \frac{1}{2}\epsilon\boldsymbol{c}(S(G, \boldsymbol{r})) = \frac{1}{2}\epsilon\text{OPT}$. Finally, by Theorem 4.6, the cost of the output is at most $(1 + c\epsilon)$ OPT. This gives:

Theorem 6.2. *There is an approximation scheme for solving the $\{0, 1, 2\}$-edge connectivity problem (allowing duplication of edges) in planar graphs. The running time is $O(2^{\text{poly}(1/\epsilon)}n + n\log n)$.*

Comments The PTAS framework used is potentially applicable to problems where (i) the input consists of a planar graph G with edge-costs and a subset Q of the vertices of G (we call Q the set of *terminals*), and where (ii) the output spans the terminals. Steiner tree and two-edge-connectivity have been solved using this framework. The PTAS for the subset tour problem [19] (which was the inspiration for this framework) can be reframed using this technique. Since the extended abstract of this work first appeared, Borradaile, Demaine and Tazari have also this framework to give PTASes for the same set of problems in graphs of bounded genus [6], Bateni, Hajiaghayi and Marx [3] have extended the framework to the Steiner forest problem and Bateni et al. [2] have extended the framework to prize collecting problems.

Acknowledgements The authors thank David Pritchard for comments on early versions of this work. This material is based upon work supported by the National Science Foundation under Grant Nos. CCF-0964037 and CCF-0963921 and by a Natural Science and Research Council of Canada Postdoctoral Fellowship.

References

[1] B. Baker. Approximation algorithms for NP-complete problems on planar graphs. *Journal of the ACM*, 41(1):153–180, 1994.

[2] M. Bateni, C. Chekuri, A. Ene, M. Hajiaghayi, N. Korula, and D. Marx. Prize-collecting Steiner problems on planar graphs. In *Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1028–1049, 2011.

[3] M. Bateni, M. Hajiaghayi, and D. Marx. Approximation schemes for Steiner forest on planar graphs and graphs of bounded treewidth. *J. ACM*, 58(5):21, 2011.

[4] A. Berger, A. Czumaj, M. Grigni, and H. Zhao. Approximation schemes for minimum 2-connected spanning subgraphs in weighted planar graphs. In *Proceedings of the 13th European Symposium on Algorithms*, volume 3669 of *Lecture Notes in Computer Science*, pages 472–483, 2005.

[5] A. Berger and M. Grigni. Minimum weight 2-edge-connected spanning subgraphs in planar graphs. In *Proceedings of the 34th International Colloquium on Automata, Languages and Programming*, volume 4596 of *Lecture Notes in Computer Science*, pages 90–101, 2007.

[6] G. Borradaile, E. Demaine, and S. Tazari. Polynomial-time approximation schemes for subset-connectivity problems in bounded-genus graphs. *Algorithmica*, 2012. Online.

[7] G. Borradaile, C. Kenyon-Mathieu, and P. Klein. A polynomial-time approximation scheme for Steiner tree in planar graphs. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1285–1294, 2007.

[8] G. Borradaile, P. Klein, and C. Mathieu. Steiner tree in planar graphs: An $O(n \log n)$ approximation scheme with singly exponential dependence on epsilon. In *Proceedings of the 10th International Workshop on Algorithms and Data Structures*, volume 4619 of *Lecture Notes in Computer Science*, pages 275–286, 2007.

[9] G. Borradaile, P. Klein, and C. Mathieu. An $O(n \log n)$ approximation scheme for Steiner tree in planar graphs. *ACM Transactions on Algorithms*, 5(3):1–31, 2009.

[10] A. Czumaj and A. Lingas. On approximability of the minimum cost k-connected spanning subgraph problem. In *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 281–290, 1999.

[11] R. Erickson, C. Monma, and A. Veinott. Send-and-split method for minimum-concave-cost network flows. *Mathematics of Operations Research*, 12:634–664, 1987.

[12] K. Eswaran and R. Tarjan. Augmentation problems. *SIAM Journal on Computing*, 5(4):653–665, 1976.

[13] G. Frederickson and J. Jájá. Approximation algorithms for several graph augmentation problems. *SIAM Journal on Computing*, 10(2):270–283, 1981.

[14] M. Goemans, A. Goldberg, S. Plotkin, D. Shmoys, É. Tardos, and D. Williamson. Improved approximation algorithms for network design problems. In *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 223–232, 1994.

[15] M. Henzinger, P. Klein, S. Rao, and S. Subramanian. Faster shortest-path algorithms for planar graphs. *Journal of Computer and System Sciences*, 55(1):3–23, 1997.

[16] K. Jain. A factor 2 approximation algorithm for the generalized Steiner network problem. *Combinatorica*, 2001(1):39–60, 21.

[17] R. Jothi, B. Raghavachari, and S. Varadarajan. A 5/4-approximation algorithm for minimum 2-edge-connectivity. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 725–734, 2003.

[18] S. Khuller and U. Vishkin. Biconnectivity approximations and graph carvings. *Journal of the ACM*, 41(2):214–235, 1994.

[19] P. Klein. A subset spanner for planar graphs, with application to subset TSP. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing*, pages 749–756, 2006.

[20] P. Klein. A linear-time approximation scheme for TSP in undirected planar graphs with edge-weights. *SIAM Journal on Computing*, 37(6):1926–1952, 2008.

[21] P. Klein and R. Ravi. When cycles collapse: A general approximation technique for constraind two-connectivity problems. In *Proceedings of the 3rd International Conference on Integer Programming and Combinatorial Optimization*, pages 39–55, 1993.

[22] K. Mehlhorn. A faster approximation algorithm for the Steiner problem in graphs. *Information Processing Letters*, 27(3):125–128, 1988.

[23] S. Nakano and T. Uno. Efficient generation of rooted trees. Technical Report NII-2003-005E, National Institute of Informatics, 2003.

[24] R. Ravi. Approximation algorithms for Steiner augmentations for two-connectivity. Technical Report TR-CS-92-21, Brown University, 1992.

[25] M. Resende and P. Pardalos, editors. *Handbook of Optimization in Telecommunications*. Springer, 2006.

[26] P. Widmayer. A fast approximation algorithm for Steiner's problem in graphs. In *Graph-Theoretic Concepts in Computer Science*, volume 246 of *Lecture Notes in Computer Science*, pages 17–28. Springer Verlag, 1986.

[27] D. Williamson, M. Goemans, M. Mihail, and V. Vazirani. A primal-dual approximation algorithm for generalized Steiner network problems. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, pages 708–717, 1993.

[28] Y. Wu, P. Widmayer, and C. Wong. A faster approximation algorithm for the Steiner problem in graphs. *Acta informatica*, 23(2):223–229, 1986.